
cepton_sdk Documentation

Cepton Technologies

Sep 12, 2019

CONTENTS

1	Overview	1
2	Samples	3
3	Errors	17
4	Setup	21
5	Sensors	25
6	Points	29
7	Serial	31
8	Networking	33
9	Capture Replay	35
10	C++	37
11	Utilities	39
	Index	45

OVERVIEW

The Cepton SDK provides the following features

- **Parsing:** parse sensor packets
- **Calibration:** apply sensor calibration
- **Networking:** listen for sensor packets
- **Capture Replay:** read sensor packets from a PCAP file
- **Frame Accumulation:** accumulate and output sensor points by frame

Currently, the Cepton LiDAR packet formats are under active development, and are not publicly available. The SDK is required for **Parsing** and **Calibration**. All other SDK features are optional, and can be done manually by the user.

1.1 Getting Started

To start, take a look at [Samples](#). The sample code covers most common use cases.

For prototyping, it is recommended to use the high level C++ API ([C++](#), [Utilities](#)).

1.2 Timestamps

All `int64` timestamps are microseconds since the Unix epoch (UTC). All `float` timestamps are time differences measured in seconds. Point timestamps are based on one of the following sources (the first valid source is used):

1. GPS (NMEA + PPS)
2. PTP
3. Host PC

All `float` and `int64` time differences are seconds (measurement period, replay time, frame length, etc.).

1.3 Multiple Returns

To enable multiple returns, pass the `CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS` flag during initialization.

The returns are as follows:

1. Strongest signal.

2. Furthest signal, if it is not the strongest. Otherwise, the second strongest signal.

1.4 Concurrency

- If networking is enabled, the sdk creates 2 threads for networking. All callbacks occur in these networking threads.
- If networking is disabled, the sdk does not create any threads. All callbacks occur in the main thread.
- All sdk getter functions are thread safe, and can be called from callbacks. Other sdk functions are not guaranteed to be thread safe, and can cause deadlock if called from callbacks.

1.5 Minimal SDK

If desired, the following SDK features can be disabled in the SDK and performed manually by the user:

- Networking: *Network*.
- Capture Replay: *Replay*.
- Frame Accumulation: *Frame*.

1.6 Sensor Fusion

Cepton sensors do not have a concept of a frame boundaries. This is especially useful with multiple sensors, since there is no need to worry about frame synchronization between sensors. We recommend choosing a desired frame length, and collecting data for all sensors simultaneously (*Process Single*, *Process Multi*). Longer frame lengths will allow for higher point density per frame, but higher latency. A frame length of ~0.1s (10Hz) is recommended for most applications.

SAMPLES

2.1 Basic

Listing 1: samples/basic.cpp

```
1  /**
2   * Sample code for general sdk usage.
3   */
4  #include <iostream>
5  #include <string>
6  #include <vector>
7
8  #undef CEPTON_ENABLE_EXCEPTIONS
9
10 #include <cepton_sdk_api.hpp>
11
12 class FramesListener {
13 public:
14     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
15                        const cepton_sdk::SensorImagePoint *c_image_points) {
16         // Get sensor info
17         cepton_sdk::SensorInformation sensor_info;
18         cepton_sdk::api::check_error(
19             cepton_sdk::get_sensor_information(handle, sensor_info));
20
21         // Print info
22         if (i_frame < 5) {
23             std::printf("Received %i points from sensor %i\n", (int)n_points,
24                        (int)sensor_info.serial_number);
25         }
26         ++i_frame;
27     }
28
29 private:
30     std::size_t i_frame = 0;
31 };
32
33 int main(int argc, char **argv) {
34     std::string capture_path;
35     if (argc >= 2) capture_path = argv[1];
36
37     // Initialize
38     auto options = cepton_sdk::create_options();
```

(continues on next page)

(continued from previous page)

```

39 options.frame.mode = CEPTON_SDK_FRAME_TIMED;
40 options.frame.length = 0.1f;
41 cepton_sdk::api::check_error(
42     cepton_sdk::api::initialize(options, capture_path));
43
44 // Get sensor
45 std::printf("Waiting for sensor to connect...\n");
46 while (cepton_sdk::get_n_sensors() == 0)
47     cepton_sdk::api::check_error(cepton_sdk::api::wait(0.1f));
48 cepton_sdk::SensorInformation sensor_info;
49 cepton_sdk::api::check_error(
50     cepton_sdk::get_sensor_information_by_index(0, sensor_info));
51 std::printf("Sensor: %d\n", (int)sensor_info.serial_number);
52
53 // Listen for frames
54 std::printf("Listening for frames...\n");
55 cepton_sdk::api::SensorImageFrameCallback callback;
56 cepton_sdk::api::check_error(callback.initialize());
57 FramesListener frames_listener;
58 callback.listen(&frames_listener, &FramesListener::on_image_frame);
59
60 // Run
61 cepton_sdk::api::check_error(cepton_sdk::api::wait());
62
63 // Deinitialize
64 cepton_sdk::api::check_error(cepton_sdk::deinitialize());
65 }

```

2.2 C Basic

Listing 2: samples/c_basic.c

```

1  /**
2   * Sample code for general C sdk usage.
3   */
4  #include <stdio.h>
5  #include <time.h>
6
7  #include <cepton_sdk.h>
8
9  int frames_got = 0;
10 time_t first_frame_time = 0;
11 time_t current_frame_time = 0;
12
13 void image_frame_callback(CeptonSensorHandle handle, size_t n_points,
14                          const struct CeptonSensorImagePoint *c_points,
15                          void *user_data) {
16     time_t t = time(NULL);
17
18     if (frames_got == 0) first_frame_time = t;
19     frames_got++;
20     if (frames_got < 50)
21         printf("Frame: %4d Time: %lld\n", frames_got, (long long)t);
22     else

```

(continues on next page)

(continued from previous page)

```

23     printf("Frame: %4d Time: %lld Frame rate: %.1fHz\n", frames_got,
24           (long long)t, frames_got * 1.0 / (t - first_frame_time));
25 }
26
27 int main() {
28     printf("Start\n");
29     fflush(stdout);
30     struct CeptonSDKOptions options = cepton_sdk_create_options();
31     options.frame.mode = CEPTON_SDK_FRAME_TIMED;
32     options.frame.length = 0.1f;
33     CeptonSensorErrorCode ret;
34     ret = cepton_sdk_initialize(CEPTON_SDK_VERSION, &options, NULL, NULL);
35     if (ret != CEPTON_SUCCESS) printf("%s\n", cepton_get_error_code_name(ret));
36     ret = cepton_sdk_listen_image_frames(image_frame_callback, NULL);
37     if (ret != CEPTON_SUCCESS) printf("%s\n", cepton_get_error_code_name(ret));
38
39     while (frames_got < 100)
40         ; // Just spin loop for lack of cross platform sleep in C
41
42     cepton_sdk_deinitialize();
43
44     return 0;
45 }

```

2.3 Callback

Listing 3: samples/callback.cpp

```

1  /**
2   * Sample code for callback usage.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  // Sample global callback.
7  void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
8                    const cepton_sdk::SensorImagePoint *c_image_points) {}
9
10 // Sample member callback.
11 class FramesListener {
12 public:
13     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
14                       const cepton_sdk::SensorImagePoint *c_image_points) {}
15 };
16
17 int main(int argc, char **argv) {
18     // Initialize
19     cepton_sdk::api::check_error(cepton_sdk::api::initialize());
20     cepton_sdk::api::SensorImageFrameCallback callback;
21     cepton_sdk::api::check_error(callback.initialize());
22
23     // Listen lambda
24     callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
25                      const cepton_sdk::SensorImagePoint *c_image_points) {});
26 }

```

(continues on next page)

(continued from previous page)

```
27 // Listen global function
28 callback.listen(on_image_frame);
29
30 // Listen member function
31 FramesListener frames_listener;
32 callback.listen(&frames_listener, &FramesListener::on_image_frame);
33 }
```

2.4 Error

Listing 4: samples/error.cpp

```
1 /**
2  * Sample code for error callback usage.
3  */
4 #include <cepton_sdk_api.hpp>
5
6 int main(int argc, char** argv) {
7     // Initialize
8     cepton_sdk::api::SensorErrorCallback error_callback;
9     error_callback.listen(
10         [&](cepton_sdk::SensorHandle handle,
11             const cepton_sdk::SensorError& error) { throw error; });
12     cepton_sdk::api::check_error(cepton_sdk::initialize(
13         CEPTON_SDK_VERSION, cepton_sdk::create_options(),
14         error_callback.global_on_callback, &error_callback));
15 }
```

2.5 Sensor

Listing 5: samples/sensor.cpp

```
1 /**
2  * Sample code for sensor information.
3  */
4 #include <string>
5
6 #include <cepton_sdk_api.hpp>
7
8 int main(int argc, char **argv) {
9     std::string capture_path;
10     if (argc >= 2) capture_path = argv[1];
11
12     // Initialize
13     auto options = cepton_sdk::create_options();
14     cepton_sdk::api::check_error(
15         cepton_sdk::api::initialize(options, capture_path));
16     cepton_sdk::api::check_error(cepton_sdk::api::wait(5.0f));
17
18     // Get all sensors
19     const int n_sensors = (int) cepton_sdk::get_n_sensors();
```

(continues on next page)

(continued from previous page)

```

20  for (int i = 0; i < n_sensors; ++i) {
21      cepton_sdk::SensorInformation sensor_info;
22      cepton_sdk::api::check_error(
23          cepton_sdk::get_sensor_information_by_index(i, sensor_info));
24      std::printf("%i: %s\n", (int)sensor_info.serial_number,
25                  sensor_info.model_name);
26  }
27  }

```

2.6 Advanced

2.6.1 Frame

Listing 6: samples/advanced/frame.cpp

```

1  /**
2   * Sample code for custom frame accumulation.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  int main(int argc, char **argv) {
7      std::string capture_path;
8      if (argc >= 2) capture_path = argv[1];
9
10     // Initialize
11     auto options = cepton_sdk::create_options();
12     cepton_sdk::api::check_error(
13         cepton_sdk::api::initialize(options, capture_path));
14     cepton_sdk::api::SensorImageFrameCallback callback;
15     cepton_sdk::api::check_error(callback.initialize());
16
17     // Get sensor
18     while (cepton_sdk::get_n_sensors() == 0)
19         cepton_sdk::api::check_error(cepton_sdk::api::wait(0.1f));
20     cepton_sdk::SensorInformation sensor_info;
21     cepton_sdk::api::check_error(
22         cepton_sdk::get_sensor_information_by_index(0, sensor_info));
23
24     // Create accumulator
25     auto frame_options = cepton_sdk::create_frame_options();
26     frame_options.mode = CEPTON_SDK_FRAME_TIMED;
27     frame_options.length = 0.1f;
28     cepton_sdk::util::FrameAccumulator accumulator(sensor_info);
29     cepton_sdk::api::check_error(accumulator.set_options(frame_options));
30     callback.listen(
31         [&](cepton_sdk::SensorHandle handle, std::size_t n_points,
32             const cepton_sdk::SensorImagePoint *const c_image_points) {
33             if (handle != sensor_info.handle) return;
34             accumulator.add_points((int)n_points, c_image_points);
35         });
36
37     // Listen
38     accumulator.callback.listen(

```

(continues on next page)

(continued from previous page)

```

39     [&](int n_points,
40         const cepton_sdk::SensorImagePoint *const c_image_points) {
41         std::printf("Received %i points\n", n_points);
42     });
43
44     cepton_sdk::api::check_error(cepton_sdk::api::wait(5.0f));
45 }

```

2.6.2 Network

Listing 7: samples/advanced/network.cpp

```

1  /**
2   * Sample code for custom networking.
3   */
4  #include <asio.hpp>
5
6  #include <cepton_sdk_api.hpp>
7
8  using asio::ip::udp;
9
10 class SocketListener {
11 public:
12     SocketListener() : m_socket(m_io_service, udp::v4()) {
13         m_socket.set_option(asio::socket_base::reuse_address(true));
14         m_socket.bind(udp::endpoint(udp::v4(), 8808));
15     }
16
17     void run() {
18         listen();
19         m_io_service.run_for(std::chrono::seconds(5));
20     }
21
22     void listen() {
23         m_socket.async_receive_from(
24             asio::buffer(m_buffer), m_end_point,
25             [this](const asio::error_code& error, std::size_t buffer_size) {
26                 if (buffer_size == 0) return;
27                 if (error == asio::error::operation_aborted) return;
28                 const CeptonSensorHandle handle =
29                     m_end_point.address().to_v4().to_ulong();
30                 // For more accurate timestamps, a separate network receive thread
31                 // should be
32                 // used.
33                 const int64_t timestamp = cepton_sdk::util::get_timestamp_usec();
34                 cepton_sdk::api::check_error(cepton_sdk::mock_network_receive(
35                     handle, timestamp, m_buffer.data(), buffer_size));
36                 listen();
37             });
38     }
39
40 private:
41     asio::io_context m_io_service;
42     udp::socket m_socket;
43     udp::endpoint m_end_point;

```

(continues on next page)

(continued from previous page)

```

44     std::array<uint8_t, 4096> m_buffer;
45 };
46
47 int main() {
48     // Initialize sdk
49     auto options = cepton_sdk::create_options();
50     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
51     options.frame.mode = CEPTON_SDK_FRAME_COVER;
52     cepton_sdk::api::check_error(cepton_sdk::api::initialize(options));
53
54     // Listen for points
55     cepton_sdk::api::SensorImageFrameCallback callback;
56     cepton_sdk::api::check_error(callback.initialize());
57     callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
58                     const cepton_sdk::SensorImagePoint* c_image_points) {
59         std::printf("Received %i points from sensor %lli\n", (int)n_points,
60                     (long long)handle);
61     });
62
63     SocketListener listener;
64     listener.run();
65 }

```

2.6.3 Organize Points

Listing 8: samples/advanced/organize_points.cpp

```

1  /*****
2  **
3  ** Copyright(C) 2019 Cepton Technologies. All Rights Reserved.
4  ** Contact: https://www.cepton.com
5  **
6  ** Sample code which opens a cepton sensor pcap file, organizes
7  ** the points and continuously saves the most recent organized
8  ** points to a frame to a cvs file "organized_cloud.cvs
9  *****/
10
11 #include <cepton_sdk_util.hpp>
12 #include <cepton_sdk/capture.hpp>
13 #include <cepton_sdk_api.hpp>
14
15 using namespace cepton_sdk::util;
16
17 int main(int argc, char** argv) {
18     if (argc < 2) return -1;
19     const std::string capture_path = argv[1];
20
21     // Initialize sdk
22     auto options = cepton_sdk::create_options();
23     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
24     options.frame.mode = CEPTON_SDK_FRAME_COVER;
25     cepton_sdk::api::check_error(cepton_sdk::api::initialize(options));
26
27     cepton_sdk::SensorInformation sensor_info;
28

```

(continues on next page)

(continued from previous page)

```

29   OrganizedCloud organized_cloud;
30   std::ofstream os;
31
32   cepton_sdk::Capture m_capture;
33   cepton_sdk::api::check_error(m_capture.open_for_read(capture_path));
34
35   cepton_sdk::api::check_error(
36       cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
37                                   CEPTON_SDK_CONTROL_DISABLE_NETWORK));
38   cepton_sdk::api::check_error(cepton_sdk_clear());
39
40   // Listen for points
41   cepton_sdk::api::SensorImageFrameCallback callback;
42   cepton_sdk::api::check_error(callback.initialize());
43   callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
44                       const cepton_sdk::SensorImagePoint* c_image_points) {
45
46       cepton_sdk::get_sensor_information(handle, sensor_info);
47
48       std::printf("Received %i points from sensor %lli\n", static_cast<int>(n_points),
49                   static_cast<long long>(handle));
50
51       Organizer organizer(sensor_info);
52
53       organizer.organize_points(n_points,
54                                sensor_info.return_count,
55                                c_image_points,
56                                organized_cloud);
57
58       os.open("organize_cloud.csv");
59       for (const auto& point : organized_cloud.points)
60       {
61           if (point.valid)
62           {
63               float x = 0;
64               float y = 0;
65               float z = 0;
66               cepton_sdk::util::convert_image_point_to_point(
67                   point.image_x, point.image_z, point.distance, x,
68                   y, z);
69
70               os << x << ", " << y << ", " << z << "\n";
71           }
72       }
73       os.close();
74   });
75
76   while (true) {
77       cepton_sdk::Capture::PacketHeader header;
78       const uint8_t* data;
79       cepton_sdk::api::check_error(m_capture.next_packet(header, data));
80
81       const cepton_sdk::SensorHandle handle =
82           static_cast<cepton_sdk::SensorHandle>(header.ip_v4) |
83           CEPTON_SENSOR_HANDLE_FLAG MOCK;
84       cepton_sdk::api::check_error(cepton_sdk_mock_network_receive(
85           handle, header.timestamp, data, static_cast<size_t>(header.data_size)));

```

(continues on next page)

(continued from previous page)

```

86     }
87 }

```

2.6.4 Process Multi

Listing 9: samples/advanced/process_multi.cpp

```

1  /**
2   * Sample code for processing multiple sensor data.
3   */
4  #include <string>
5  #include <vector>
6
7  #include <cepton_sdk_api.hpp>
8
9  struct Frame {
10     int64_t timestamp;
11     std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
12         image_points_dict;
13 };
14
15 class FrameAccumulator {
16 public:
17     void on_image_frame(
18         cepton_sdk::SensorHandle handle, std::size_t n_points,
19         const cepton_sdk::SensorImagePoint* const c_image_points) {
20         cepton_sdk::util::LockGuard lock(m_mutex);
21
22         auto& image_points = m_image_points_dict[handle];
23         image_points.reserve(image_points.size() + n_points);
24         image_points.insert(image_points.end(), c_image_points,
25                             c_image_points + n_points);
26
27         check_and_publish();
28     }
29
30 private:
31     void check_and_publish() {
32         const auto timestamp = cepton_sdk::api::get_time();
33
34         if (timestamp < m_timestamp) return;
35         if ((timestamp - m_timestamp) < int64_t(m_frame_length * 1e6f)) return;
36         m_timestamp = timestamp;
37
38         auto frame = std::make_shared<Frame>();
39         frame->timestamp = timestamp;
40         frame->image_points_dict = m_image_points_dict;
41         m_image_points_dict.clear();
42         queue.push(frame);
43     }
44
45 public:
46     cepton_sdk::util::SingleConsumerQueue<Frame> queue;
47
48 private:

```

(continues on next page)

(continued from previous page)

```

49     std::timed_mutex m_mutex;
50     float m_frame_length = 0.1f;
51     int64_t m_timestamp = 0;
52     std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
53         m_image_points_dict;
54 };
55
56 int main(int argc, char** argv) {
57     std::string capture_path;
58     if (argc >= 2) capture_path = argv[1];
59
60     auto options = cepton_sdk::create_options();
61     cepton_sdk::api::check_error(
62         cepton_sdk::api::initialize(options, capture_path));
63     cepton_sdk::api::SensorImageFrameCallback callback;
64     cepton_sdk::api::check_error(callback.initialize());
65     if (cepton_sdk::capture_replay::is_open())
66         cepton_sdk::api::check_error(cepton_sdk::capture_replay::resume());
67
68     FrameAccumulator accumulator;
69     callback.listen(&accumulator, &FrameAccumulator::on_image_frame);
70
71     while (true) {
72         const auto frame = accumulator.queue.pop(0.01f);
73         if (!frame) continue;
74         // Do processing
75     }
76 }

```

2.6.5 Process Single

Listing 10: samples/advanced/process_single.cpp

```

1  /**
2   * Sample code for processing single sensor data.
3   */
4  #include <memory>
5  #include <vector>
6
7  #include <cepton_sdk_api.hpp>
8
9  struct Frame {
10     int64_t timestamp;
11     cepton_sdk::SensorHandle handle;
12     std::vector<cepton_sdk::SensorImagePoint> image_points;
13 };
14
15 int main(int argc, char **argv) {
16     std::string capture_path;
17     if (argc >= 2) capture_path = argv[1];
18
19     auto options = cepton_sdk::create_options();
20     options.frame.mode = CEPTON_SDK_FRAME_TIMED;
21     options.frame.length = 0.1f;
22     cepton_sdk::api::check_error(

```

(continues on next page)

(continued from previous page)

```

23     cepton_sdk::api::initialize(options, capture_path));
24 cepton_sdk::api::SensorImageFrameCallback callback;
25 cepton_sdk::api::check_error(callback.initialize());
26 if (cepton_sdk::capture_replay::is_open())
27     cepton_sdk::api::check_error(cepton_sdk::capture_replay::resume());
28
29 cepton_sdk::util::SingleConsumerQueue<Frame> queue;
30 callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
31                  const cepton_sdk::SensorImagePoint *c_image_points) {
32     auto frame = std::make_shared<Frame>();
33     frame->timestamp = cepton_sdk::api::get_time();
34     frame->handle = handle;
35     frame->image_points.reserve(n_points);
36     frame->image_points.insert(frame->image_points.end(), c_image_points,
37                             c_image_points + n_points);
38     queue.push(frame);
39 });
40
41 while (true) {
42     const auto frame = queue.pop(0.01f);
43     if (!frame) continue;
44     // Do processing
45 }
46 }

```

2.6.6 Replay

Listing 11: samples/advanced/replay.cpp

```

1  /**
2   * Sample code for custom packet replaying.
3   */
4  #include <cepton_sdk/capture.hpp>
5  #include <cepton_sdk_api.hpp>
6
7  class CaptureReplay {
8  public:
9      CaptureReplay(const std::string& path) {
10         cepton_sdk::api::check_error(m_capture.open_for_read(path));
11
12         cepton_sdk::api::check_error(
13             cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
14                                         CEPTON_SDK_CONTROL_DISABLE_NETWORK));
15         cepton_sdk::api::check_error(cepton_sdk_clear());
16     }
17
18     ~CaptureReplay() {
19         m_capture.close();
20         if (cepton_sdk_is_initialized()) {
21             cepton_sdk::api::check_error(cepton_sdk_clear());
22         }
23     }
24
25     void run() {
26         while (true) {

```

(continues on next page)

(continued from previous page)

```

27     cepton_sdk::Capture::PacketHeader header;
28     const uint8_t* data;
29     cepton_sdk::api::check_error(m_capture.next_packet(header, data));
30
31     const cepton_sdk::SensorHandle handle =
32         (cepton_sdk::SensorHandle)header.ip_v4 |
33         CEPTON_SENSOR_HANDLE_FLAG MOCK;
34     cepton_sdk::api::check_error(cepton_sdk_mock_network_receive(
35         handle, header.timestamp, data, header.data_size));
36     }
37 }
38
39 private:
40     cepton_sdk::Capture m_capture;
41 };
42
43 int main(int argc, char** argv) {
44     if (argc < 2) return -1;
45     const std::string capture_path = argv[1];
46
47     // Initialize sdk
48     auto options = cepton_sdk::create_options();
49     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
50     options.frame.mode = CEPTON_SDK_FRAME_COVER;
51     cepton_sdk::api::check_error(cepton_sdk::api::initialize(options));
52
53     // Listen for points
54     cepton_sdk::api::SensorImageFrameCallback callback;
55     cepton_sdk::api::check_error(callback.initialize());
56     callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
57         const cepton_sdk::SensorImagePoint* c_image_points) {
58         std::printf("Received %i points from sensor %lli\n", (int)n_points,
59             (long long)handle);
60     });
61
62     CaptureReplay replay(capture_path);
63     replay.run();
64 }

```

2.6.7 Stray

Listing 12: samples/advanced/stray.cpp

```

1  /**
2   * Sample code for stray filter usage.
3   */
4  #include <vector>
5
6  #include <cepton_sdk_api.hpp>
7
8  int main(int argc, char **argv) {
9      std::string capture_path;
10     if (argc >= 2) capture_path = argv[1];
11
12     // Initialize

```

(continues on next page)

(continued from previous page)

```

13  auto options = cepton_sdk::create_options();
14  cepton_sdk::api::check_error(
15      cepton_sdk::api::initialize(options, capture_path));
16
17  cepton_sdk::api::SensorImageFrameCallback callback;
18  cepton_sdk::api::check_error(callback.initialize());
19  cepton_sdk::util::StrayFilter filter;
20  callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
21                      const cepton_sdk::SensorImagePoint *c_image_points) {
22      // Get sensor
23      cepton_sdk::SensorInformation sensor_info;
24      cepton_sdk::api::check_error(
25          cepton_sdk::get_sensor_information(handle, sensor_info));
26
27      // Copy points to buffer
28      std::vector<cepton_sdk::SensorImagePoint> image_points;
29      image_points.insert(image_points.begin(), c_image_points,
30                          c_image_points + n_points);
31
32      // Filter stray
33      filter.init(sensor_info);
34      filter.run((int)n_points, image_points.data());
35  });
36
37  // Run
38  cepton_sdk::api::check_error(cepton_sdk::api::wait(5.0f));
39  }

```

2.6.8 Transform

Listing 13: samples/advanced/transform.cpp

```

1  /**
2   * Sample code for transforming 3d points.
3   */
4  #include "cepton_sdk_api.hpp"
5
6  int main() {
7      // Create transform
8      std::array<float, 3> translation = {0.0f, 0.0f, 1.0f};
9      std::array<float, 4> rotation = {1.0f, 0.0f, 0.0f, 0.0f};
10     auto compiled_transform = cepton_sdk::util::CompiledTransform::create(
11         translation.data(), rotation.data());
12
13     // Apply transform
14     cepton_sdk::util::SensorPoint point = {};
15     compiled_transform.apply(point.x, point.y, point.z);
16     printf("Point: [%f, %f, %f]\n", point.x, point.y, point.z);
17 }

```


ERRORS

3.1 Types

class SensorError : public runtime_error

Error returned by most functions.

Implicitly convertible from/to SensorErrorCode. Getter functions do not return an error, because they cannot fail. Will call CEPTON_RUNTIME_ASSERT if nonzero error is not used (call `ignore` to manually use error).

Public Functions

SensorError (SensorErrorCode *code_*, **const** std::string &*msg_*)

SensorError (SensorErrorCode *code_*)

SensorError ()

~SensorError ()

SensorError (**const** *SensorError* &*other*)

SensorError &**operator=** (**const** *SensorError* &*other*)

bool **used** () **const**
Internal use only.

const *SensorError* &**ignore** () **const**
Mark error as used.

const std::string &**msg** () **const**
Returns error message;.

SensorErrorCode **code** () **const**
Returns error code.

operator SensorErrorCode () **const**
Implicitly convert to SensorErrorCode.

operator bool () **const**
Returns `false` if code is CEPTON_SUCCESS, true otherwise.

const std::string **name** () **const**

bool **is_error** () **const**

```
bool is_fault () const
_CeptonSensorErrorCode
    Values:
    0
    1
    2
    4
    5
    6
    Networking error.
    7
    8
    Invalid value or uninitialized struct.
    9
    10
    11
    12
    13
    14
    15
    1000
    Internal parameter out of range.
    1001
    Reading exceed spec.
    1002
    Reading exceeds spec.
    1003
    1004
    1005
    1006
    1007
    1008
typedef int32_t CeptonSensorErrorCode
```

3.2 Methods

```
const char* cepton_get_error_code_name (CeptonSensorErrorCode error_code)
    Returns string name of error code.
    Returns empty string if error code is invalid.
```

int **cepton_is_error_code** (*CeptonSensorErrorCode* error_code)

Returns true if error name is of the form CEPTON_ERROR_*, false otherwise.

int **cepton_is_fault_code** (*CeptonSensorErrorCode* error_code)

Returns true if error name is of the form CEPTON_FAULT_*, false otherwise.

CeptonSensorErrorCode **cepton_sdk_get_error** (const char ** error_msg)

Returns and clears last sdk error.

error_msg is owned by the SDK, and is valid until the next call in the current thread.

4.1 Types

`_CeptonSDKControl`

SDK control flags.

Values:

- 1
Disable networking operations.
Useful for running multiple instances of sdk in different processes. Must pass packets manually to `cepton_sdk::mock_network_receive`.
- 2
Disable marking image clipped points as invalid.
Does not affect number of points returned.
- 3
Disable marking distance clipped points as invalid.
Does not affect number of points returned.
- 4
Enable multiple returns.
When set, `cepton_sdk::SensorInformation::return_count` will indicate the number of returns per laser. Can only be set at sdk initialization.
- 5
Enable marking stray points as invalid (measurement noise).
Uses `cepton_sdk::util::StrayFilter` to mark points invalid.
Does not affect number of points returned.
- 6
Always use packet timestamps (disable GPS/PTP timestamps).
- 7
Enable marking crosstalk points as invalid.

`typedef uint32_t CeptonSDKControl`

`_CeptonSDKFrameMode`

Controls frequency of points being reported.

Values:

- 0 Report points by packet.
- 1 Report points at fixed time intervals.
Interval controlled by *CeptonSDKFrameOptions::length*.
- 2 Report points when the field of view is covered once.
 - For HR80 series, detects half scan cycle (left-to-right or right-to-left).
- 3 Report points when the scan pattern goes through a full cycle.
Typically 2x longer frame than COVER mode.
 - For HR80 series, detects full scan cycle (left-to-right-to-left).
 - For VISTA series, internally uses TIMED mode.

```
typedef uint32_t CeptonSDKFrameMode
struct CeptonSDKFrameOptions
```

Public Members

size_t **signature**
Internal use only.

CeptonSDKFrameMode **mode**
Default: CEPTON_SDK_FRAME_STREAMING.

float **length**
Frame length [seconds].
Default: 0.05. Only used if mode=CEPTON_SDK_FRAME_TIMED.

struct *CeptonSDKFrameOptions* **cepton_sdk_create_frame_options()**
Create default frame options.

struct CeptonSDKOptions
SDK initialization options.

Public Members

size_t **signature**
Internal use only.

CeptonSDKControl **control_flags**
Default: 0.

struct *CeptonSDKFrameOptions* **frame**

uint16_t **port**
Default: 8808.

struct *CeptonSDKOptions* **cepton_sdk_create_options()**
Create default options.

```
typedef void (*cepton_sdk::FpSensorErrorCallback) (SensorHandle handle, SensorErrorCode
error_code, const char *error_msg,
const void *error_data, size_t error_data_size, void *user_data)
```

4.2 Methods

CeptonSensorErrorCode **cepton_sdk_initialize** (int ver, **const** struct *CeptonSDKOptions* ***const options**, FpCeptonSensorErrorCallback cb, void ***const user_data**)

Initializes settings and networking.

Must be called before any other sdk function listed below.

CeptonSensorErrorCode **cepton_sdk_deinitialize** ()
Resets everything and deallocates memory.

CeptonSensorErrorCode **cepton_sdk_clear** ()
Clears sensors.

Use when loading/unloading capture file.

CeptonSensorErrorCode **cepton_sdk_set_control_flags** (*CeptonSDKControl* mask, *CeptonSDKControl* flags)

CeptonSDKControl **cepton_sdk_get_control_flags** ()

int **cepton_sdk_has_control_flag** (*CeptonSDKControl* flag)

uint16_t **cepton_sdk_get_port** ()

CeptonSensorErrorCode **cepton_sdk_set_port** (uint16_t port)
Sets network listen port.

Default: 8808.

CeptonSensorErrorCode **cepton_sdk_set_frame_options** (**const** struct *CeptonSDKFrameOptions* ***const options**)

CeptonSDKFrameMode **cepton_sdk_get_frame_mode** ()

float **cepton_sdk_get_frame_length** ()

SENSORS

5.1 Types

```
typedef CeptonSensorHandle cepton_sdk::SensorHandle
```

```
_CeptonSensorModel
```

Values:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

15

```
typedef uint16_t CeptonSensorModel
```

```
struct CeptonSensorInformation
```

Public Members

CeptonSensorHandle **handle**

uint64_t **serial_number**

char CeptonSensorInformation::model_name[28]

```
CeptonSensorModel model
uint16_t reserved
char CeptonSensorInformation::firmware_version[28]
uint8_t major
uint8_t minor
uint8_t CeptonSensorInformation::unused[2]
struct CeptonSensorInformation::@16 formal_firmware_version
float last_reported_temperature
    [celsius]
float last_reported_humidity
    [%]
float last_reported_age
    [hours]
float measurement_period
    Time between measurements [seconds].
int64_t ptp_ts
    [microseconds]
uint8_t gps_ts_year
    0-99 (2017 -> 17)
uint8_t gps_ts_month
    1-12
uint8_t gps_ts_day
    1-31
uint8_t gps_ts_hour
    0-23
uint8_t gps_ts_min
    0-59
uint8_t gps_ts_sec
    0-59
uint8_t return_count
uint8_t segment_count
    Number of image segments.
uint32_t flags
    Bit flags.
uint32_t is_mocked
    Created by capture replay.
uint32_t is_pps_connected
    GPS PPS is available.
uint32_t is_nmea_connected
    GPS NMEA is available.
uint32_t is_ptp_connected
    PTP is available.
```

`uint32_t is_calibrated`

`uint32_t is_over_heated`

Hit temperature limit (only available in Vista Gen2 for now)

`uint32_t is_sync_firing_enabled`

`union CeptonSensorInformation::@17 CeptonSensorInformation::@18`

5.2 Methods

`size_t cepton_sdk_get_n_sensors ()`

Get number of sensors attached. Use to check for new sensors. Sensors are not deleted until deinitialization.

`CeptonSensorErrorCode cepton_sdk_get_sensor_handle_by_serial_number (uint64_t serial_number,
CeptonSensorHandle
*const handle)`

Returns error if sensor not found.

`CeptonSensorErrorCode cepton_sdk_get_sensor_information_by_index (size_t idx, struct
CeptonSensorInformation *const info)`

Valid indices are in range [0, n_sensors). Returns error if index invalid.

`CeptonSensorErrorCode cepton_sdk_get_sensor_information (CeptonSensorHandle handle,
struct CeptonSensorInformation
*const info)`

Returns error if sensor not found.

6.1 Types

struct CeptonSensorImagePoint

Point in image coordinates (focal length = 1).

To convert to 3d point, refer to `cepton_sdk_util.hpp`.

Public Members

`int64_t timestamp`

Unix time [microseconds].

`float image_x`

x image coordinate.

`float distance`

Distance [meters].

`float image_z`

z image coordinate.

`float intensity`

Diffuse reflectance.

`CeptonSensorReturnType return_type`

`uint8_t flags`

Bit flags.

`uint8_t valid`

If `false`, then the distance and intensity are invalid.

`uint8_t saturated`

If `true`, then the intensity is invalid. Also, the distance is valid, but inaccurate.

`union CeptonSensorImagePoint::@21 CeptonSensorImagePoint::@22`

`uint8_t CeptonSensorImagePoint::reserved[2]`

6.2 Methods

```
typedef void (* FpCeptonSensorImageDataCallback) (CeptonSensorHandle handle,
                                                    size_t n_points, const struct Cep-
                                                    tonSensorImagePoint *c_points,
                                                    void *user_data)
```

Callback for receiving image points.

Set the frame length to control the callback rate.

```
CeptonSensorErrorCode cepton_sdk_listen_image_frames (FpCeptonSensorImageDataCallback cb,
                                                    void *const user_data)
```

Sets image frames callback.

Returns points at frequency specified by `cepton_sdk::FrameOptions::mode`. Each frame contains all possible points (use `cepton_sdk::SensorImagePoint::valid` to filter points). Points are ordered by measurement, segment, and return:

```
measurement_count = n_points / (segment_count * return_count)
idx = ((i_measurement) * segment_count + i_segment) * return_count + i_return
```

Returns error if callback already registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_image_frames ()
```

SERIAL

Serial callback. Primarily used for receiving data from GPS/INS attached to sensor.

7.1 Types

```
typedef void (* FpCeptonSerialReceiveCallback) (CeptonSensorHandle handle, const  
char *str, void *user_data)  
    Callback for receiving serial data (e.g. NMEA).
```

7.2 Methods

```
CeptonSensorErrorCode cepton_sdk_listen_serial_lines (FpCeptonSerialReceiveCallback cb,  
void *const user_data)
```

Sets serial line callback.

Useful for listening to NMEA data from GPS attached to sensor.

Each callback contains 1 line of serial data (including newline characters).

Returns error if callback already registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_serial_lines ()
```


NETWORKING

Network callback for debugging.

8.1 Types

```
typedef void (* FpCeptonNetworkReceiveCallback) (CeptonSensorHandle handle,  
int64_t timestamp, const uint8_t *buffer,  
size_t buffer_size, void *user_data)
```

Callback for receiving network packets.

Parameters

- *handle*: Unique sensor identifier (e.g. IP address). Returns error if callback already set.

8.2 Methods

```
CeptonSensorErrorCode cepton_sdk_listen_network_packet (FpCeptonNetworkReceiveCallback cb,  
void *const user_data)
```

Sets network packets callback.

Only 1 callback can be registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_network_packet ()
```


CAPTURE REPLAY

PCAP capture file replay. Source code can be found in the *source* folder. Functions are not thread safe, and should only be called from the main thread.

int **cepton_sdk_capture_replay_is_open** ()

CeptonSensorErrorCode **cepton_sdk_capture_replay_open** (const char *const *path*)

Opens capture file.

Must be called before any other replay functions listed below.

CeptonSensorErrorCode **cepton_sdk_capture_replay_close** ()

const char* **cepton_sdk_capture_replay_get_filename** ()

int64_t **cepton_sdk_capture_replay_get_start_time** ()

Returns capture start timestamp (unix time [microseconds]).

float **cepton_sdk_capture_replay_get_position** ()

Returns capture file position [seconds].

float **cepton_sdk_capture_replay_get_length** ()

Returns capture file length [seconds].

int **cepton_sdk_capture_replay_is_end** ()

Returns true if at end of capture file.

This is only relevant when using *resume_blocking* methods.

CeptonSensorErrorCode **cepton_sdk_capture_replay_seek** (float *position*)

Seek to capture file position [seconds].

Position must be in range [0.0, capture length). Returns error if position is invalid.

CeptonSensorErrorCode **cepton_sdk_capture_replay_set_enable_loop** (int *enable_loop*)

If enabled, replay will automatically rewind at end.

int **cepton_sdk_capture_replay_get_enable_loop** ()

CeptonSensorErrorCode **cepton_sdk_capture_replay_set_speed** (float *speed*)

Replay speed multiplier for asynchronous replay.

float **cepton_sdk_capture_replay_get_speed** ()

CeptonSensorErrorCode **cepton_sdk_capture_replay_resume_blocking_once** ()

Replay next packet in current thread without sleeping.

Pauses replay thread if it is running.

CeptonSensorErrorCode **cepton_sdk_capture_replay_resume_blocking** (float *duration*)

Replay multiple packets synchronously.

No sleep between packets. Resume duration must be non-negative. Pauses replay thread if it is running.

int **cepton_sdk_capture_replay_is_running** ()

Returns true if replay thread is running.

CeptonSensorErrorCode **cepton_sdk_capture_replay_resume** ()

Packets are replayed in realtime. Replay thread sleeps in between packets.

CeptonSensorErrorCode **cepton_sdk_capture_replay_pause** ()

Pauses asynchronous replay thread.

High level C++ API for prototyping (*cepton_sdk_api.hpp*). Methods are agnostic to live/replay mode.

`bool cepton_sdk::api::is_live()`
Returns true if capture replay is not open.

`bool cepton_sdk::api::is_end()`

`int64_t cepton_sdk::api::get_time()`
Returns capture replay time or live time.

SensorError `cepton_sdk::api::wait` (float *t_length* = -1.0f)
Sleeps or resumes capture replay for duration.
If *t_length* < 0, then waits forever.

10.1 Errors

`const SensorError &cepton_sdk::api::check_error` (const *SensorError* &*error*, const std::string &*msg* = "")

Handles error.

If error, raises exception. Otherwise, prints error.

`const SensorError &cepton_sdk::api::log_error` (const *SensorError* &*error*, const std::string &*msg* = "")

Prints error.

`void cepton_sdk::api::default_on_error` (*SensorHandle* *h*, *SensorErrorCode* *error_code*, const char *const *error_msg*, const void *const *error_data*, std::size_t *error_data_size*, void *const *instance*)

Basic SDK error callback.

Calls `cepton_sdk::api::check_error_code`.

10.2 Setup

SensorError `cepton_sdk::api::initialize` (Options *options* = `create_options()`, const std::string &*capture_path* = "")

Initialize SDK and optionally starts capture replay.

`class SensorErrorCallback : public cepton_sdk::util::Callback<SensorHandle, const SensorError&>`
Callback for sensor errors.

Public Static Functions

```
static void global_on_callback(SensorHandle handle, SensorErrorCode error_code, const
                             char *error_msg, const void *const error_data, size_t er-
                             ror_data_size, void *const instance)
```

class SensorImageFrameCallback : public cepton_sdk::util::Callback<*SensorHandle*, std::size_t, const SensorImagePo
 Callback for image frames.
 Must call `initialize` before use.

Public Functions

```
~SensorImageFrameCallback()
```

```
SensorError initialize()
```

```
SensorError deinitialize()
```

```
bool is_initialized() const
```

class NetworkPacketCallback : public cepton_sdk::util::Callback<*SensorHandle*, int64_t, uint8_t const *, std::size_t>
 Callback for network packets.
 Must call `initialize` before use.

Public Functions

```
~NetworkPacketCallback()
```

```
SensorError initialize()
```

```
SensorError deinitialize()
```

10.3 Sensors

```
bool cepton_sdk::api::has_sensor_by_serial_number(uint64_t serial_number)
```

```
SensorError cepton_sdk::api::get_sensor_information_by_serial_number(uint64_t se-
                                                                    rial_number,
                                                                    SensorIn-
                                                                    formation
                                                                    &info)
```

Returns error if sensor not found.

```
std::vector<uint64_t> cepton_sdk::api::get_sensor_serial_numbers()
```

Returns serial numbers for all sensors.

UTILITIES

Utility functions and classes for prototyping (*cepton_sdk_util.hpp*).

11.1 Common

`int64_t cepton_sdk::util::get_timestamp_usec()`

Returns current unix timestamp [microseconds].

This is the timestamp format used by all sdk functions.

11.2 Points

`void cepton_sdk::util::convert_image_point_to_point` (float *image_x*, float *image_z*, float *distance*, float &*x*, float &*y*, float &*z*)

Convert image point to 3d point.

struct SensorPoint

3d point class.

Can't subclass from SensorImagePoint, needs to be POD.

Public Members

`int64_t timestamp`
Unix time [microseconds].

`float image_x`
x image coordinate.

`float distance`
Distance [meters].

`float image_z`
z image coordinate.

`float intensity`
Diffuse reflectance.

`CeptonSensorReturnType return_type`

`uint8_t flags`

```

uint8_t valid
uint8_t saturated
union cepton_sdk::util::SensorPoint::[anonymous] [anonymous]
uint8_t reserved[5]
float x
    x cartesian coordinate
float y
    y cartesian coordinate
float z
    z cartesian coordinate
void cepton_sdk::util::convert_sensor_image_point_to_point (const    SensorImage-
                                                             Point    &image_point,
                                                             SensorPoint &point)
Convenience    method    to    convert    cepton_sdk::SensorImagePoint    to
cepton_sdk::SensorPoint.

```

11.3 Transforms

class CompiledTransform

3d translation and rotation.

For more functionality, use Eigen's Geometry module.

Public Functions

void **apply** (float &x, float &y, float &z)
 Apply transformation to 3d position.

Public Static Functions

static *CompiledTransform* **create** (const float *const *translation*, const float *const *rotation*)
 Create from translation and rotation.

Parameters

- *translation*: Cartesian (x, y, z)
- *rotation*: Quaternion (x, y, z, w)

11.4 Callbacks

template<typename ...**TArgs**>

class Callback

Expands SDK callback functionality.

Allows for multiple callbacks to be registered. Allows for registering lambdas and member functions. See `samples/basic.cpp`.

Public Functions

```
void clear ()
    Clear all listeners.

SensorError listen (const std::function<void> TArgs...
    > &func, uint64_t *const id = nullptr) Register std::function.

template<typename TClass>
SensorError listen (TClass *const instance, MemberFunction<TClass, TArgs...> func, uint64_t
    *const id = nullptr)
    Register instance member function.

void operator () (TArgs... args) const
    Emit callback.
```

Public Static Functions

```
static void global_on_callback (TArgs... args, void *const instance)
    Used for registering as c callback.
```

11.5 Frames

class FrameDetector

Detects frames in streaming sensor data.

Public Functions

```
FrameDetector (const SensorInformation &sensor_info)

const FrameOptions &get_options () const

SensorError set_options (const FrameOptions &options)

void reset ()
    Completely resets detector.

    Only use if also clearing points accumulator.

bool add_point (const SensorImagePoint &point)
    Returns true if frame found.

    Automatically resets after frame is found.
```

Public Members

```
bool frame_found

int frame_idx

float frame_x
    Number of points in current frame.
```

class FrameAccumulator

Accumulates image points, and emits frames to callback.

See `samples/frame.cpp`.

Public Functions

FrameAccumulator (**const** SensorInformation &*sensor_info*)

FrameOptions **get_options** () **const**

SensorError **set_options** (**const** FrameOptions &*options*)

void **clear** ()

void **add_points** (int *n_points*, **const** SensorImagePoint ***const** *image_points*)

Public Members

Callback<int, **const** SensorImagePoint *> **callback**

11.6 Organizer

struct OrganizedCloud

The *OrganizedCloud* struct An organized version of the cepton point cloud.

Public Functions

int **getIndex** (int *row*, int *col*, int *n_return*)

getIndex Returns the index of the point corresponding to the inputted row, col and return number.

Return**Parameters**

- [in] *row*: Row index
- [in] *col*: Col index
- [in] *n_return*: Return index

Public Members

int64_t **timestamp_start**

timestamp_start The time of the oldest point in the cloud

int64_t **timestamp_end**

timestamp_end The time of the newest point in the cloud

int **height**

height Height of the cloud. Represents how many rows there are in the cloud

int **width**

width Width of the cloud. Represents how many columns there are in the cloud

```
int n_returns
    n_returns Number of return represented by the cloud.

std::vector<CellInfo> info_cells
    info_cells Vector of cell info which provide information about the matching points

std::vector<CeptonSensorImagePoint> points
    points Vector of organized points. Stored in Return, Row, Col order. So to get a point at row 10, col 15,
    return 1 would be points[(row * width
        • col) n_returns + return
```

```
struct CellInfo
    The CellInfo struct.
```

Public Members

```
bool occupied_cell = false
    occupied_cell Is the cell at this index occupied with a point. If false can't assume this represents free
    space.

int original_index = -1
    original_index Index of the point that was used to generate the organized point. Can be used to match
    back with original data if required. Should only be use if occupied_cell is true.
```

class Organizer

The *Organizer* class Performs organization on cepton unorganized points. Creates an angular grid, places each point within that grid and outputs a point for each location in the grid in a row/col format. Thread safe. Defaults to a 0.4deg spaced grid.

Public Types

enum OrganizerMode

Values:

RECENT

Output the most recent point from the frame that fell within the grid

CENTER

Output the center of the grid. Uses median point distance.

Public Functions

```
Organizer (cepton_sdk::SensorInformation sensor_info)
    Organizer.
```

Parameters

- *sensor_info*: Sensor info for organizer. Used to set min/max angles

```
void organize_points (const int num_points_in, const int n_returns, const CeptonSensorImagePoint *const unorganized_points, cepton_sdk::util::OrganizedCloud &organized_points)

    organize_points
```

Parameters

- [in] `num_points_in`: Number of unorganized points
- [in] `n_returns`: Number of returns
- [in] `unorganized_points`: Unorganized points to process
- [out] `organized_points`: Points in organized form

void **mode** (*OrganizerMode* mode)
mode

Parameters

- mode: Change the mode of the organizer. [RECENT] Points are the most recent which fill within the grid. [CENTER] Points outputted are at the center of the grid. More even spacing but less accurate.

void **binSize** (float *bin_size*)
binSize Change the bin size of the organizer

Parameters

- bin_size: The horizontal and vertical bin size to set. In radians

void **settings** (*OrganizerSettings* organizer_settings)
settings

Parameters

- organizer_settings: Change organizer settings

Organizer::OrganizerSettings **settings** ()
settings

Return The settings the organizer is using

struct OrganizerSettings

Public Members

float **horizontal_range_radians** = to_radians(70.f)

float **vertical_range_radians** = to_radians(30.f)

float **horizontal_bin_size_radians** = to_radians(0.4f)

float **vertical_bin_size_radians** = to_radians(0.4f)

OrganizerMode **mode** = *OrganizerMode::*RECENT

C

- cepton_get_error_code_name (*C function*), 18
- cepton_is_error_code (*C function*), 18
- cepton_is_fault_code (*C function*), 19
- cepton_sdk::api::check_error (*C++ function*), 37
- cepton_sdk::api::default_on_error (*C++ function*), 37
- cepton_sdk::api::get_sensor_information_by_serial_number (*C++ function*), 38
- cepton_sdk::api::get_sensor_serial_numbers (*C++ function*), 38
- cepton_sdk::api::get_time (*C++ function*), 37
- cepton_sdk::api::has_sensor_by_serial_number (*C++ function*), 38
- cepton_sdk::api::initialize (*C++ function*), 37
- cepton_sdk::api::is_end (*C++ function*), 37
- cepton_sdk::api::is_live (*C++ function*), 37
- cepton_sdk::api::log_error (*C++ function*), 37
- cepton_sdk::api::NetworkPacketCallback (*C++ class*), 38
- cepton_sdk::api::NetworkPacketCallback::NetworkPacketCallback (*C++ function*), 38
- cepton_sdk::api::NetworkPacketCallback::define_initialize (*C++ function*), 38
- cepton_sdk::api::NetworkPacketCallback::initialize (*C++ function*), 38
- cepton_sdk::api::SensorErrorCallback (*C++ class*), 37
- cepton_sdk::api::SensorErrorCallback::global_on_callback (*C++ function*), 38
- cepton_sdk::api::SensorImageFrameCallback (*C++ class*), 38
- cepton_sdk::api::SensorImageFrameCallback::SensorImageFrameCallback (*C++ function*), 38
- cepton_sdk::api::SensorImageFrameCallback::define_initialize (*C++ function*), 38
- cepton_sdk::api::SensorImageFrameCallback::initialize (*C++ function*), 38
- cepton_sdk::api::SensorImageFrameCallback::is_initialized (*C++ function*), 38
- cepton_sdk::api::wait (*C++ function*), 37
- cepton_sdk::FpSensorErrorCallback (*C++ type*), 22
- cepton_sdk::SensorError (*C++ class*), 17
- cepton_sdk::SensorError::~SensorError (*C++ function*), 17
- cepton_sdk::SensorError::code (*C++ function*), 17
- cepton_sdk::SensorError::ignore (*C++ function*), 17
- cepton_sdk::SensorError::is_error (*C++ function*), 17
- cepton_sdk::SensorError::is_fault (*C++ function*), 17
- cepton_sdk::SensorError::msg (*C++ function*), 17
- cepton_sdk::SensorError::name (*C++ function*), 17
- cepton_sdk::SensorError::operator bool (*C++ function*), 17
- cepton_sdk::SensorError::operator SensorErrorCode (*C++ function*), 17
- cepton_sdk::SensorError::operator= (*C++ function*), 17
- cepton_sdk::SensorError::SensorError (*C++ function*), 17
- cepton_sdk::SensorError::used (*C++ function*), 17
- cepton_sdk::SensorHandle (*C++ type*), 25
- cepton_sdk::util::Callback (*C++ class*), 40
- cepton_sdk::util::Callback::clear (*C++ function*), 41
- cepton_sdk::util::Callback::global_on_callback (*C++ function*), 41
- cepton_sdk::util::Callback::listen (*C++ function*), 41
- cepton_sdk::util::Callback::operator () (*C++ function*), 41
- cepton_sdk::util::CompiledTransform (*C++ class*), 40
- cepton_sdk::util::CompiledTransform::apply (*C++ function*), 40

(C++ function), 40	(C++ member), 43
cepton_sdk::util::CompiledTransform::create	cepton_sdk::util::OrganizedCloud::n_returns
(C++ function), 40	(C++ member), 42
cepton_sdk::util::convert_image_point_to_cp	cepton_sdk::util::OrganizedCloud::points
(C++ function), 39	(C++ member), 43
cepton_sdk::util::convert_sensor_image_po	cepton_sdk::util::OrganizedCloud::timestamp_end
(C++ function), 40	(C++ member), 42
cepton_sdk::util::FrameAccumulator (C++	cepton_sdk::util::OrganizedCloud::timestamp_start
class), 41	(C++ member), 42
cepton_sdk::util::FrameAccumulator::add_p	cepton_sdk::util::OrganizedCloud::width
(C++ function), 42	(C++ member), 42
cepton_sdk::util::FrameAccumulator::call	cepton_sdk::util::Organizer (C++ class), 43
(C++ member), 42	cepton_sdk::util::Organizer::binSize
cepton_sdk::util::FrameAccumulator::clear	(C++ function), 44
(C++ function), 42	cepton_sdk::util::Organizer::CENTER
cepton_sdk::util::FrameAccumulator::FrameAccu	(C++ enumerator), 43
(C++ function), 42	cepton_sdk::util::Organizer::mode (C++
cepton_sdk::util::FrameAccumulator::get_option	function), 44
(C++ function), 42	cepton_sdk::util::Organizer::organize_points
cepton_sdk::util::FrameAccumulator::set_option	(C++ function), 43
(C++ function), 42	cepton_sdk::util::Organizer::Organizer
cepton_sdk::util::FrameDetector (C++	(C++ function), 43
class), 41	cepton_sdk::util::Organizer::OrganizerMode
cepton_sdk::util::FrameDetector::add_point	(C++ enum), 43
(C++ function), 41	cepton_sdk::util::Organizer::OrganizerSettings
cepton_sdk::util::FrameDetector::frame_found	(C++ class), 44
(C++ member), 41	cepton_sdk::util::Organizer::OrganizerSettings::hor
cepton_sdk::util::FrameDetector::frame_idx	(C++ member), 44
(C++ member), 41	cepton_sdk::util::Organizer::OrganizerSettings::hor
cepton_sdk::util::FrameDetector::frame_x	(C++ member), 44
(C++ member), 41	cepton_sdk::util::Organizer::OrganizerSettings::mod
cepton_sdk::util::FrameDetector::FrameDetector	(C++ member), 44
(C++ function), 41	cepton_sdk::util::Organizer::OrganizerSettings::ver
cepton_sdk::util::FrameDetector::get_options	(C++ member), 44
(C++ function), 41	cepton_sdk::util::Organizer::OrganizerSettings::ver
cepton_sdk::util::FrameDetector::reset	(C++ member), 44
(C++ function), 41	cepton_sdk::util::Organizer::RECENT
cepton_sdk::util::FrameDetector::set_options	(C++ enumerator), 43
(C++ function), 41	cepton_sdk::util::Organizer::settings
cepton_sdk::util::get_timestamp_usec	(C++ function), 44
(C++ function), 39	cepton_sdk::util::SensorPoint (C++ class),
cepton_sdk::util::OrganizedCloud (C++	39
class), 42	cepton_sdk::util::SensorPoint::distance
cepton_sdk::util::OrganizedCloud::CellInfo	(C++ member), 39
(C++ class), 43	cepton_sdk::util::SensorPoint::flags
cepton_sdk::util::OrganizedCloud::CellInfo::occ	(C++ member), 39
(C++ member), 43	cepton_sdk::util::SensorPoint::image_x
cepton_sdk::util::OrganizedCloud::CellInfo::ori	(C++ member), 39
(C++ member), 43	cepton_sdk::util::SensorPoint::image_z
cepton_sdk::util::OrganizedCloud::getIndex	(C++ member), 39
(C++ function), 42	cepton_sdk::util::SensorPoint::intensity
cepton_sdk::util::OrganizedCloud::height	(C++ member), 39
(C++ member), 42	cepton_sdk::util::SensorPoint::reserved
cepton_sdk::util::OrganizedCloud::info_cells	(C++ member), 40

cepton_sdk::util::SensorPoint::return_type *tion*), 22
 (C++ member), 39 cepton_sdk_create_options (C function), 22
 cepton_sdk::util::SensorPoint::saturated cepton_sdk_deinitialize (C function), 23
 (C++ member), 40 cepton_sdk_get_control_flags (C function),
 cepton_sdk::util::SensorPoint::timestamp 23
 (C++ member), 39 cepton_sdk_get_error (C function), 19
 cepton_sdk::util::SensorPoint::valid cepton_sdk_get_frame_length (C function), 23
 (C++ member), 39 cepton_sdk_get_frame_mode (C function), 23
 cepton_sdk::util::SensorPoint::x (C++ cepton_sdk_get_n_sensors (C function), 27
 member), 40 cepton_sdk_get_port (C function), 23
 cepton_sdk::util::SensorPoint::y (C++ cepton_sdk_get_sensor_handle_by_serial_number
 member), 40 (C function), 27
 cepton_sdk::util::SensorPoint::z (C++ cepton_sdk_get_sensor_information (C
 member), 40 function), 27
 cepton_sdk::util::SensorPoint::[anonymous] cepton_sdk_get_sensor_information_by_index
 (C++ member), 40 (C function), 27
 cepton_sdk_capture_replay_close (C func- cepton_sdk_has_control_flag (C function), 23
 tion), 35 cepton_sdk_initialize (C function), 23
 cepton_sdk_capture_replay_get_enable_loo cepton_sdk_listen_image_frames (C func-
 (C function), 35 tion), 30
 cepton_sdk_capture_replay_get_filename cepton_sdk_listen_network_packet (C func-
 (C function), 35 tion), 33
 cepton_sdk_capture_replay_get_length (C cepton_sdk_listen_serial_lines (C func-
 function), 35 tion), 31
 cepton_sdk_capture_replay_get_position cepton_sdk_set_control_flags (C function),
 (C function), 35 23
 cepton_sdk_capture_replay_get_speed (C cepton_sdk_set_frame_options (C function),
 function), 35 23
 cepton_sdk_capture_replay_get_start_time cepton_sdk_set_port (C function), 23
 (C function), 35 cepton_sdk_unlisten_image_frames (C func-
 cepton_sdk_capture_replay_is_end (C func- tion), 30
 tion), 35 cepton_sdk_unlisten_network_packet (C
 cepton_sdk_capture_replay_is_open (C function), 33
 function), 35 cepton_sdk_unlisten_serial_lines (C func-
 cepton_sdk_capture_replay_is_running (C tion), 31
 function), 36
 cepton_sdk_capture_replay_open (C func- tion), 35
 tion), 35
 cepton_sdk_capture_replay_pause (C func- tion), 36
 tion), 36
 cepton_sdk_capture_replay_resume (C func- tion), 36
 tion), 36
 cepton_sdk_capture_replay_resume_blocking (C function), 35
 (C function), 35
 cepton_sdk_capture_replay_resume_blocking_once (C function), 35
 (C function), 35
 cepton_sdk_capture_replay_seek (C func- tion), 35
 tion), 35
 cepton_sdk_capture_replay_set_enable_loop (C function), 35
 (C function), 35
 cepton_sdk_capture_replay_set_speed (C function), 35
 function), 35
 cepton_sdk_clear (C function), 23
 cepton_sdk_create_frame_options (C func-